УДК 681.3.06

ИСПОЛЬЗОВАНИЕ НАБОРА ДИАГРАММ UML ДЛЯ ПОСТРОЕНИЯ МОДЕЛЕЙ ПРОИЗВОДИТЕЛЬНОСТИ

С.А. Дубаков, В.А. Силич

Томский политехнический университет
E-mail: sad@osu.cctpu.edu.ru, vas@osu.cctpu.edu.ru

Рассматривается возможность генерации моделей производительности программного обеспечения на основе диаграмм в нотации UML как одна из базовых составляющих методологии интеграции анализа производительности в процесс разработки. Предложен подход, основанный на методологии Software Performance Engineering (SPE), использующий в качестве исходных данных стандартные элементы UML и ряд расширений.

Основной причиной выявляемых проблем производительности программного обеспечения (ПО) является инерционный подход к данной проблеме на протяжении процесса разработки. Данный подход, именуемый также в литературе как «fix-it-later» («исправим-это-потом»), заключается в том, что вопросы производительности игнорируются до того момента, когда выявляются проблемы. В случае если это происходит (а происходит это практически всегда, за редким исключением), приходится принимать решение об увеличении аппаратных ресурсов, модифицировании исходного кода или перепроектировании системы или ее частей. Подобные затраты часто приводят к срывам сроков и превышению бюджета, а в некоторых случаях достижение требуемой производительности в рамках текущего проекта становится практически невозможным.

Проблемы с производительностью в большинстве случаев возникают вследствие неверных архитектурных решений, а не неэффективного кодирования. Это означает, что они привносятся в проект на ранних этапах разработки, но оказываются невыявленными до момента интеграционного тестирования или передачи системы в эксплуатацию, когда решить их становится гораздо сложнее.

Альтернативу рассмотренному типичному подходу составляет превентивный подход к вопросу производительности программного обеспечения. Это означает, что производительность рассматривается и учитывается на всех этапах жизненного цикла наряду с остальными ключевыми характеристиками программного обеспечения. Все важные проектные и административные решения должны приниматься в том числе с учетом вопросов производительности. Оценка производительности должна выполняться тем или иным способом при рассмотрении критических архитектурных альтернатив, а менеджмент должен учитывать затраты на рассмотрение и решение вопросов производительности при выделении объемов работ и составлении планов.

Одним из наиболее критичных вопросов при интеграции анализа производительности в процесс разработки программного обеспечения является возможность использования уже полученных артефактов разработки в качестве исходных данных для анализа.

В настоящее время, когда объектно-ориентированный подход стал использоваться повсеместно при проектировании и реализации сложных программных систем, стандартом моделирования дефакто стал Unified Modeling Language (UML). Появившись в 1997 г., UML объединил в себе все лучшее, что было наработано в области моделирования объектно-ориентированного программного обеспечения. UML – открытый стандарт, контролируемый некоммерческим консорциумом OMG (Object Management Group) [1].

Таким образом, очевидно, что для того, чтобы как можно более тесно интегрировать анализ производи-

тельности в процесс проектирования программного обеспечения, необходимо отталкиваться от моделей ПО, специфицированных с помощью UML [2].

К настоящему моменту опубликован ряд работ, посвященных вопросам моделирования производительности на основе диаграмм UML. В большинстве случаев авторы ограничиваются формулированием идей и направлений исследования, а не предложением и реализацией полноценных решений [3–6].

Создательница Software Performance Engineering (SPE), Конни Смит [7], а также ряд других авторов [8, 9] рассматривают в своих работах перспективы моделирования производительности на основе методологии SPE [10, 11] с одной стороны и диаграмм UML в качестве исходных данных с другой. Предложенные подходы достаточно сильно отличаются. Если в [8] делается попытка построения как модели выполнения программы (на основе диаграмм вариантов использования и последовательности), так и модели выполнения системы (на основе диаграмм развертывания), то авторы [9] ограничиваются рассмотрением только первой модели.

Ряд статей [9, 12] затрагивает вопросы расширения нотации UML для включения в диаграммы информации, необходимой для моделирования производительности, в частности временные параметры и требования к ресурсам. Для этого предлагается использовать такие механизмы, как стереотипы (Stereotype), помеченные значения (Tagged value) и ограничения (Constraint).

Необходимо отметить, что ни в одной из известных авторам статей не представлено полноценного решения проблемы автоматической генерации адекватной модели производительности на основе набора связанных диаграмм UML.

Более того, некоторые из предлагаемых методик являются достаточно спорными. В частности, рассматриваемое в одной из работ [9] построение комбинированных диаграмм с помощью включения диаграмм состояний внутрь диаграмм кооперации, не выдерживает никакой критики с точки зрения практической реализации, поскольку помимо достаточно сложных алгоритмов анализа для автоматической генерации модели производительности, требует больших трудозатрат на стадии создания самих диаграмм UML для выявления и фиксации всех связей между ними.

Таким образом, существует необходимость создания полноценной методики генерации модели производительности на основе набора диаграмм UML с использованием опыта, накопленного другими авторами в этом вопросе. При этом должны быть устранены недостатки и пробелы существующих методик, а также сделан акцент на возможности автоматической трансформации модели UML в модель производительности.

Как и в ряде уже упомянутых подходов [7–9], предлагается взять за основу методологию Software Performance Engineering. В соответствии с ней, для моделирования производительности необходимы две основные модели: выполнения программы и выполнения системы.

Модель выполнения программы должна содержать сценарии рабочей нагрузки для каждой существующей функции системы. Список существующих в системе функций можно получить из модели UML, воспользовавшись диаграммами вариантов использования. Пример диаграммы с шестью вариантами использования и одним актером (User) приведен на рис. 1.

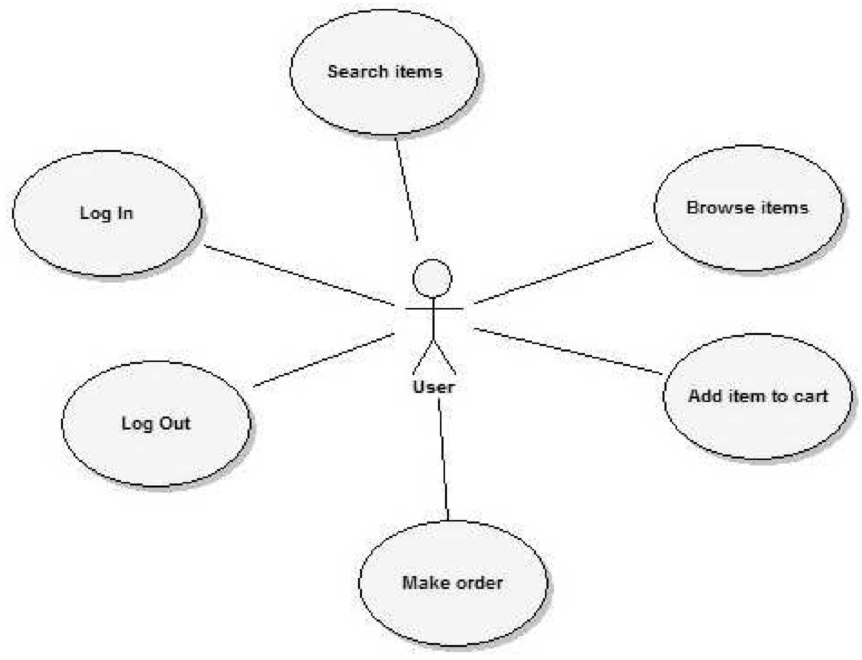


Рис. 1. *Диаграмма вариантов использования*

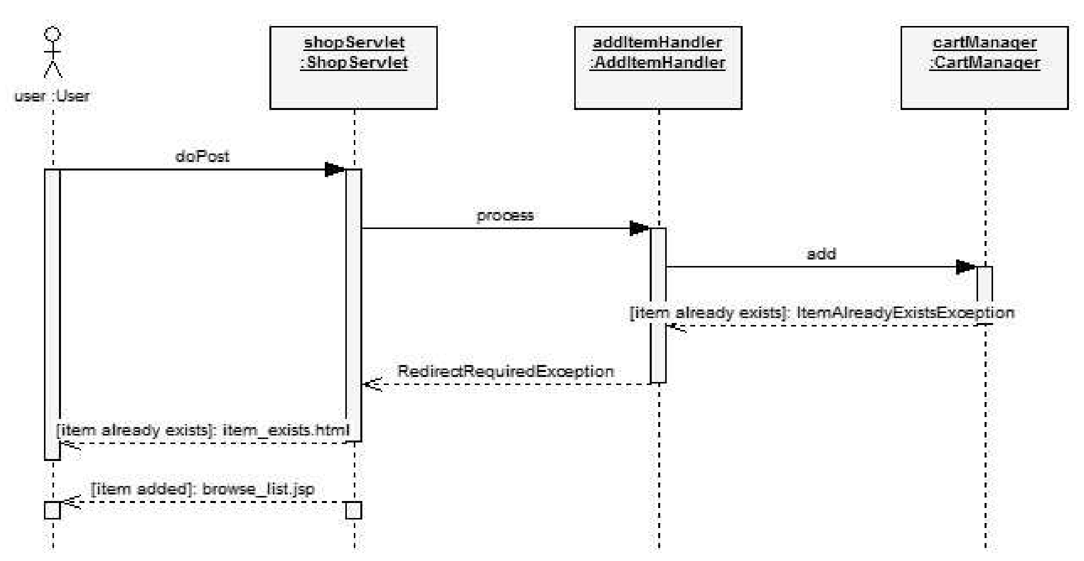


Рис. 2. *Диаграмма последовательности*

Варианты использования должны быть детализированы с помощью диаграмм последовательности, которые и являются основой для создания диаграмм выполнения SPE, описывающих сценарии рабочей нагрузки. В случае наличия альтернативных путей выполнения, диаграмм последовательности, ассоциированных с вариантом использования, может быть несколько. Пример диаграммы последовательности одного из вариантов использования приведен на рис. 2.

Этих двух диаграмм достаточно для моделирования рабочей нагрузки с аналитической точки зрения без учета особенностей физической реализации системы. Однако, наиболее ценным в процессе анализа производительности является именно возможность рассмотрения различных альтернативных архитектурных решений. Для этого модель выполнения программы должна быть дополнена деталями реализации, позволяющими оценить требования к аппаратным ресурсам окружения системы.

Такую информацию можно извлечь из диаграмм классов и диаграмм компонент. При этом объектами диаграмм взаимодействия должны быть классы, детализируемые с помощью диаграмм классов. Диаграмма классов, содержащая классы, объекты которых участвуют в диаграмме последовательности, изображенной на рис. 2, представлена на рис. 3.

Классы, в свою очередь, должны быть сгруппированы в компоненты на диаграммах компонентов. Компоненты являются связующим звеном между моделью выполнения программы и модели выполнения системы, поскольку с одной стороны они группируют классы реализации, участвующие в сценариях рабочей нагрузки, а с другой – привязываются к аппаратным ресурсам окружения. Диаграмма компонентов, группирующая классы, изображенные на рис. 3, приведена на рис. 4.

Аппаратные ресурсы окружения системы изображаются на диаграмме развертывания. Каждый узел диаграммы представляет собой некоторый тип вычислительного устройства. Соединения между узлами показывают коммуникационные каналы, с помощью которых осуществляются системные взаимодействия.

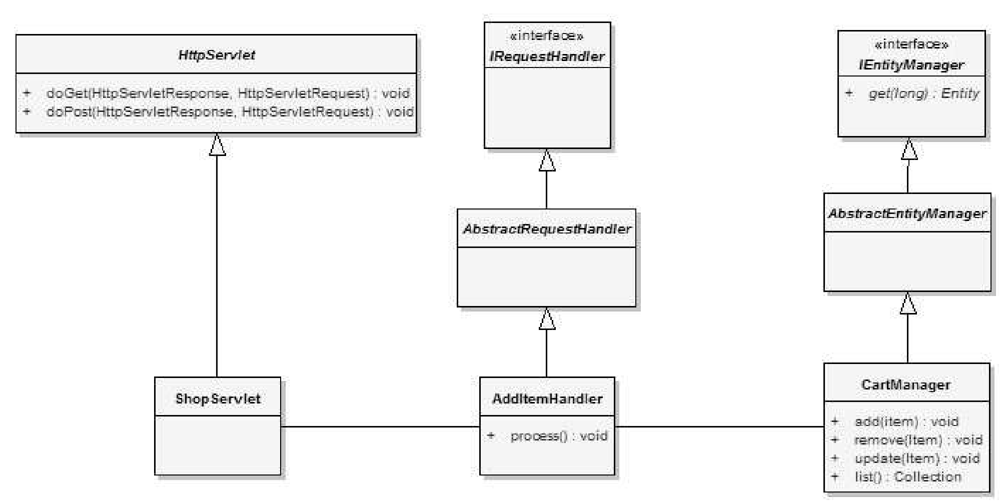


Рис. 3. *Диаграмма классов*

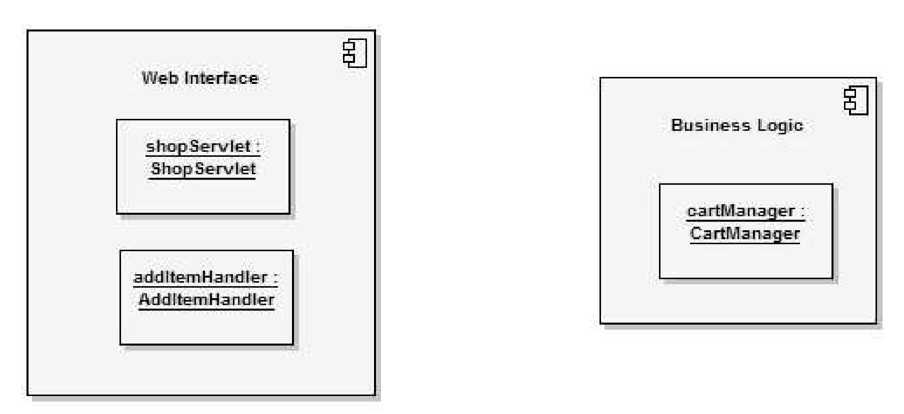


Рис. 4. *Диаграмма компонентов*

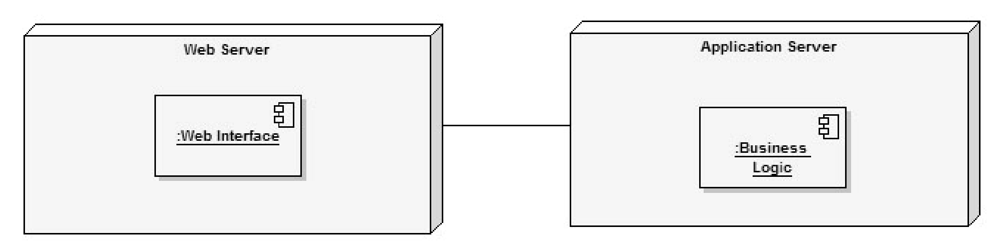


Рис. 5. *Диаграмма развертывания*

Диаграмма развертывания содержит информацию о размещении компонентов, входящих в диаграмму компонентов, в узлах системы, образуя таким образом последнее звено цепи, начинающейся с диаграмм вариантов использования, и продолжающейся последовательно с помощью диаграмм последовательности, классов и компонентов. Таким образом, этот вид диаграмм содержит необходимые исходные данные для построения модели выполнения системы. Пример диаграммы развертывания приведен на рис. 5.

Перечисленных диаграмм в стандартной нотации UML достаточно для построения как модели выполнения программы, так и модели выполнения системы, однако в них отсутствуют данные, позволяющие наполнить полученную модель параметрами производительности.

Для решения этой проблемы предлагается воспользоваться механизмом помеченных значений (tagged values). Они позволяют добавлять метаданные к элементам моделей UML и изображаются на диаграммах в виде строки «название параметра = значение», заключенной в фигурные скобки.

С помощью помеченных значений модель UML должна быть дополнена следующими данными, требуемыми для наполнения модели производительности системы:

- Диаграммы классов:
  - Для операций классов должна быть указана относительная вычислительная сложность операции – помеченное значение «complexity».
- Диаграммы развертывания:
  - Для узлов системы необходимо определить вычислительную мощность – помеченное значение «capacity».
  - Для ассоциаций между узлами, представляющими собой коммуникационные каналы, должна быть специфицирована пропускная способность – помеченное значение «bandwidth».

Получив на основе диаграмм UML, расширенных таким образом, данные, необходимые для построения модели выполнения программы и модели выполнения системы, последующую генерацию и анализ модели производительности можно осуществлять, используя один из таких математических формализмов, как обобщенные стохастические сети Петри [13], многоуровневые сети массового обслуживания [14], алгебра процессов [15] или с помощью имитационного моделирования [16].

СПИСОК ЛИТЕРАТУРЫ

1. Fowler M. UML Distilled. Third Edition. – Boston: Addison Wesley, 2003. – 192 p.
2. Дубаков С.А., Силич В.А. Моделирование производительности программного обеспечения на основе диаграмм UML // Актуальные проблемы информатики и информационных технологий: Матер. Междунар. (VIII Тамбовской межвузовской) научно-практ. конф. – Тамбов: Изд-во ТГУ им. Г.Р. Державина, 2004. – С. 52–53.
3. Gomaa H., Menasce D.A. Design and Performance Modeling of Component Interconnection Patterns for Distributed Software Architectures // ACM Proc. of Workshop on Software and Performance, WOSP2000. – Ottawa, Canada, 2000. – P. 117–126.
4. Petriu D. Deriving Performance Models from UML Models by Graph Transformations // Tutorial in ACM Proc. of Workshop on Software and Performance, WOSP2000. – Ottawa, Canada, 2000. – P. 12–19.
5. Petriu D., Wang X. From UML Descriptions of High-Level Software Architectures to LQN Performance Models // Proc. of Applications of Graph Transformations with Industrial Relevance International Workshop, AGTIVE'99. – Berlin: Springer Verlag, 1999. – P. 47–62.
6. Arief L.B., Speirs N.A. A UML Tool for an Automatic Generation of Simulation Programs // ACM Proc. of Workshop on Software and Performance, WOSP2000. – Ottawa, Canada, 2000. – P. 71–76.
7. Williams L.G., Smith C.U. Performance Evaluation of Software Architectures // Proc. of Workshop on Software and Performance, WOSP'98. – Santa Fe, New Mexico, USA, 1998. – P. 164–177.
8. Cortellessa V., Mirandola R. Deriving a Queuing Network based Performance Model from UML Diagrams // ACM Proc. of Workshop on Software and Performance, WOSP2000. – Ottawa, Canada, 2000. – P. 58–70.
9. King P., Pooley R. Derivation of Petri Net Performance Models from UML Specifications of Communications Software // Proc. 11th Intern. Conf., TOOLS 2000. – Berlin: Springer-Verlag, 2000. – P. 262–276.
10. Smith C.U. Performance Engineering of Software Systems. – Boston: Addison-Wesley, 1990. – 570 p.
11. Dubakov S. Performance Evaluation Integration into Object-Oriented Design Process // Proc. 8th Korea-Russia Intern. Symp. on Science and Technology KORUS-2004. – Tomsk, 2004. – P. 22–24.
12. De Miguel M., Lambolais T., Hannouz M., Betgй-Brezetz S., Piekarec S. UML Extensions for the Specification and Evaluation of Latency Constraints in Architectural Models // ACM Proc. of Workshop on Software and Performance, WOSP2000. – Ottawa, Canada, 2000. – P. 83–88.
13. Ajmone Marsan M., Balbo G., Conte G., Donatelli S., Franceschinis G. Modeling with Generalized Stochastic Petri Nets. – N.Y.: John Wiley and Sons, 1995. – 324 p.
14. Lavenberg S.S. A perspective on queuing models of computer performance // Performance Evaluation. – 1989. – V. 10. – № 1. – P. 53–76.
15. Hermanns H., Herzog U., Katoen J.-P. Process Algebra for Performance Evaluation // Theoretical Computer Science. – 2002. – V. 274. – № 1–2. – P. 43–87.
16. Balsamo S., Marzolla M. A Simulation-based Approach to Software Performance Modeling // Proc. 9th European software engineering conference held jointly with 11th ACM SIGSOFT Intern. Symp. on Foundations of software engineering. – N.Y.: ACM Press, 2003. – P. 363–366.